SIAM J. SCI. COMPUT.
Vol. 36, No. 6, pp. A2584–A2610

EFFICIENT LOCALIZATION OF DISCONTINUITIES IN COMPLEX COMPUTATIONAL SIMULATIONS*

ALEX GORODETSKY† AND YOUSSEF MARZOUK†

Abstract. Surrogate models for computational simulations are input-output approximations that allow computationally intensive analyses, such as uncertainty propagation and inference, to be performed efficiently. When a simulation output does not depend smoothly on its inputs, the error and convergence rate of many approximation methods deteriorate substantially. This paper details a method for efficiently localizing discontinuities in the input parameter domain, so that the model output can be approximated as a piecewise smooth function. The approach comprises an initialization phase, which uses polynomial annihilation to assign function values to different regions and thus seed an automated labeling procedure, followed by a refinement phase that adaptively updates a kernel support vector machine representation of the separating surface via active learning. The overall approach avoids structured grids and exploits any available simplicity in the geometry of the separating surface, thus reducing the number of model evaluations required to localize the discontinuity. The method is illustrated on examples of up to eleven dimensions, including algebraic models and ODE/PDE systems, and demonstrates improved scaling and efficiency over other discontinuity localization approaches.

Key words. discontinuity detection, polynomial annihilation, function approximation, support vector machines, active learning, uncertainty quantification

AMS subject classifications. 65D15, 65C20, 62K20, 62G08

DOI. 10.1137/140953137

1. Introduction.

Many applications of uncertainty quantification, optimization, and control must invoke models accessible only through computational simulation. These tasks can be computationally prohibitive, requiring repeated simulations that may exceed available computational capacity. In these circumstances, it useful to construct surrogate models approximating the simulation output over a parameter domain of interest, using a limited set of simulation runs. The construction of surrogates is essentially a problem in function approximation, for which an enormous variety of approaches have been developed. One broad category of approximations involves parametric or semiparametric representations—for instance, polynomial expansions obtained via interpolation, projection, or regression [41, 14, 42, 9, 12]. Another category involves nonparametric approximations such as Gaussian process regression [31], frequently used in the statistics community for the "emulation" of computer models [10, 32]

Almost all of these approximation methods deteriorate in efficiency when faced with discontinuities in the model output, or even its derivatives, over the range of input parameters. Yet discontinuities frequently arise in practice, e.g., when systems exhibit bifurcations with respect to uncertain input parameters or enter different regimes of operation depending on their inputs. Examples include ignition phenomena in combustion kinetics [28], bifurcations in climate modeling [39], switch-like behavior in gene expression [13], and, in general, dynamical systems with multiple equilibria. In

*Submitted to the journal's Methods and Algorithms for Scientific Computing section January 17, 2014; accepted for publication (in revised form) July 28, 2014; published electronically November 4, 2014.

http://www.siam.org/journals/sisc/36-6/95313.html

†Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139 (goroda@mit.edu, ymarz@mit.edu). The work of the first author was supported by BP.

all of these applications, being able to *localize* the discontinuity would enable significant efficiency gains in the construction of output surrogates. Moreover, localizing a discontinuity may be of standalone interest; since a discontinuous response may be a defining feature of the system, learning exactly which input or parameter regimes yield different behaviors can lead to a more fundamental understanding of the system's dynamics.

In this work, we will focus on piecewise smooth model responses; in other words, we assume that the parameter space contains one or more "separating surfaces" that bound regimes over which the model output is a smooth function of its parameters. Jumps in the model output occur across a separating surface. The separating surface may itself be relatively smooth and well approximated by techniques which take advantage of this regularity. The major contribution of this work is then an unstructured approach for identifying and refining a functional description of the separating surface. Our approach uses guided random sampling to place new model evaluation points in the vicinity of the discontinuity. These points are labeled and used to drive a kernel support vector machine classifier, which yields a nonparametric description of the discontinuity location. The entire approach is iterative: following an initialization and labeling phase, it employs a cycle of active learning, labeling, and classification. The overall algorithm uses significantly fewer model evaluations and exhibits improved scaling with parameter dimension compared to current discontinuity detection techniques. It contrasts with efforts that have generally attempted to create a dense and structured grid of model evaluations surrounding the separating surface.

The remainder of this paper is organized as follows. Section 2 reviews current techniques for discontinuity detection and for approximating discontinuous model responses. Section 3 describes the algorithmic building blocks from which we construct our approach. In section 4 we detail the discontinuity detection algorithm itself. In section 5 we report on numerical experiments with this algorithm: discontinuity detection problems of increasing dimension, problems that vary the complexity of the separating surface, and several benchmark ODE and PDE problems drawn from the literature.

2. Background. Approximation schemes for discontinuous model outputs typically attempt to transform the problem into one that can be tackled with classical approximation methods for smooth functions. These transformations can roughly be divided into three categories: local approximations, edge tracking, and global approximations.

Local approximations may involve either decomposing the parameter space in a structured manner (e.g., into hypercubes) or utilizing local basis functions. Examples of parameter space decomposition include multi-element generalized polynomial chaos [38] or treed Gaussian processes [17, 16, 4]; examples of local basis functions include wavelets [22, 21] or particular forms of basis enrichment [15]. These techniques attempt simultaneously to find the discontinuity and to build the approximation. Edge tracking techniques, on the other hand, separate discontinuity localization from approximation and concentrate on the former [18]. Another approach that separates discontinuity localization from approximation can be found in [33], where a Bayesian classification method is used to represent the separating surface, and the two resulting classes are mapped to distinct hypercubes wherein the functions are approximated by polynomial chaos expansions. Finally, there exist global methods that attempt to directly mitigate the Gibbs phenomena arising from approximating discontinuous functions with a smooth basis. One such effort [7] employs Padé–Legendre approxima-

tions in combination with filtering to remove spurious oscillations. These techniques have been successfully demonstrated in low-dimensional parameter spaces.

Below we elaborate on domain decomposition and edge tracking methods, as they provide useful inspiration for our present method.

2.1. Domain decomposition and local approximation. Decomposition techniques approach the approximation problem by breaking a domain containing a discontinuity into subdomains containing smooth portions of the function of interest. Examples are given in [38] and [1]. These algorithms may be distinguished according to three attributes: refinement criteria, point selection scheme, and approximation type. Refinement criteria are indicators that specify the need for additional function evaluations; for example, they may be tied to an estimate of the discontinuity location or to a local indicator of error in the function approximation. Point selection describes the manner in which additional function evaluations are added, e.g., deterministically or randomly, near or far from previous evaluations, etc. Finally, the approximation type may involve a choice between low- or high-order polynomials, parametric or nonparametric schemes, etc. These choices are closely intertwined because the refinement criteria and point selection scheme are often guided by the type of approximation performed in each subdomain. Many current techniques for domain decomposition rely on adaptively partitioning the parameter domain into progressively smaller hypercubes. Building approximations on these Cartesian product domains is convenient but can be computationally expensive, particularly when separating surfaces are not aligned with the coordinate axes. These difficulties are exacerbated as the parameter dimension increases.

Related to domain decomposition are approaches that use local basis functions to capture sharp variations in model output [22, 21] . These approaches also tend to rely on the progressive refinement of hypercubes. Localized bases are extensively employed in the image processing community [20]; for example, images often have sharp edges that are accurately represented with wavelets [26, 11] and other functions with local support. In practice, these basis functions are often deployed within an adaptive approach that yields a dense grid of function evaluations surrounding the discontinuity. The resulting model runs occur in similar locations and with a number/computational cost similar to domain decomposition methods.

2.2. Edge tracking. An alternative and rather efficient algorithm for discontinuity localization has been developed in [18]. As noted above, the algorithm focuses on searching for a discontinuity and developing a description of the separating surface, rather than on approximating the true model. In particular, the algorithm progressively adds points by "walking" along the discontinuity (i.e., edge tracking), while using polynomial annihilation (PA) along the coordinate axes as an indicator of the discontinuity's existence and location. This procedure uses an adaptive divide-and-conquer approach to initially locate the separating surface. After edge tracking is complete, new evaluation locations are classified—i.e., deemed to lie on one side of the separating surface or the other—using a nearest neighbor approach. The majority of the computational effort is thus spent evaluating the model near the separating surface, such that the resulting set of points becomes an evenly spaced grid surrounding it. Having located the discontinuity, function approximation can then proceed on each surrounding subdomain. For example, edge tracking is coupled with the method of least orthogonal interpolation [29] in [19].

Because a greater fraction of its computational effort is spent evaluating the model close to the separating surface, edge tracking is more efficient at discontinuity localiza-

tion than the domain decomposition methods presented earlier. The method proposed in this paper capitalizes on this philosophy and aims for further improvement by taking advantage of the regularity of the separating surface. Rather than walking along the surface with steps of fixed resolution, we introduce a new method for *sampling* in the vicinity of the discontinuity and for efficiently describing the geometry of the separating surface given an unstructured set of sample points. These developments will be detailed below.

3. Algorithmic ingredients of our approach. The new discontinuity detection algorithm described in this paper is founded on several tools common in the machine learning and spectral methods communities. These tools will be used to address three problems arising in the approximation of high-dimensional discontinuous functions. The first problem involves identifying the separating surface and estimating the jump size of the discontinuity across it. The jump size is a local measure of the difference between the function values on either side of the separating surface. We will solve this problem using PA. The solution will also provide a method for labeling function evaluations on either side of the separating surface, based upon their function value. The second problem is to find an efficient representation of the geometry of the separating surface; to this end, we will employ a nonparametric approximation using *support vector machines* (SVM). The final problem involves determining locations at which to evaluate the function in order to best refine the approximation of the separating surface. Our solution to this problem will employ *uncertainty sampling* (US) techniques.

3.1. Polynomial annihilation. PA is used in order to measure the size of a discontinuity or region of rapid change in a function. This measurement is vital for determining the region to which new function evaluations belong. Following [2], a description of one-dimensional PA is given here. The local size of the discontinuity is described in terms of the jump function evaluated at a particular location in the parameter space. Suppose that $x \in \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$. The jump function, $[f](x)$, is defined to be

$$[f](x) = f(x+) - f(x-), \tag{3.1}$$

where $f(x-) = \lim_{\Delta \to 0} f(x - \Delta)$ and $f(x+) = \lim_{\Delta \to 0} f(x + \Delta)$. Therefore, $[f](x)$ is nonzero when the function is discontinuous at x, and it is zero otherwise. The main result of PA is the approximation $L_m f$ to the jump function. This approximation has the form

$$L_m f(x) = \frac{1}{q_m(x)} \sum_{x^l \in \mathcal{S}(x)} c_l(x) f(x^l), \tag{3.2}$$

where the set $\mathcal{S}(x)$ is a "stencil" of points (x^l) around x. The coefficients (c_l) are calculated by solving the system of equations

$$\sum_{x^l \in \mathcal{S}(x)} c_l(x) p_i(x^l) = p_i^{(m)}(x), \quad i = 0 \dots m, \tag{3.3}$$

where m is the order of desired annihilation and p_i are a basis for the space of univariate polynomials of degree less than or equal to m. An explicit expression for each

c_l, derived in [2], is

$$c_l(x) = \frac{m!}{\prod_{\substack{i=0 \\ i \neq l}}^{m} (x^l - x^i)}, \quad l = 0 \dots m\,. \tag{3.4}$$

The normalization factor $q(m)$ in (3.2) is

$$q_m(x) = \sum_{x^l \in \mathcal{S}^+(x)} c_l(x), \tag{3.5}$$

where $\mathcal{S}^+(x)$ is the set $\{x^l : x^l \in \mathcal{S}(x),\, x^l > x\}$. Finally, the accuracy of this approximation is

$$L_m f(x) = \begin{cases} [f](\xi) + \mathcal{O}(h(x)) & \text{if } x^{l-1} \leq \xi, x \leq x^l, \\ \mathcal{O}(h^{\min(m,k)}(x)) & \text{if } f \in C^k(I_x) \text{ for } k > 0, \end{cases} \tag{3.6}$$

where ξ is a location at which f has a jump discontinuity, I_x is the smallest interval of points $\{x^l\}$ that contains the set $\mathcal{S}(x)$, and $h(x)$ is defined as the largest difference between neighboring points in the stencil $\mathcal{S}(x)$,

$$h(x) = \max\left\{\left|x^i - x^{i-1}\right| : x^{i-1}, x^i \in \mathcal{S}(x)\right\}. \tag{3.7}$$

A proof of (3.6) is given in [2] and is based on the residual of the Taylor series expansion around the point at which the jump function is being evaluated. Note that the expressions above rely on choosing a particular order of annihilation m; as proposed in [2], we use the *minmod* scheme to enhance the performance of PA by evaluating the jump function over a range of orders $\mathcal{M} \ni m$. We will apply the one-dimensional PA scheme along each coordinate direction in order to extend it to multiple dimensions; this process will be detailed in section 4.1.

3.2. Support vector machines. In the algorithm to be detailed in section 4, we will label function evaluations according to which side of the separating surface they lie on. An SVM [5, 35, 37], a supervised learning technique, is then used to build a boundary between the different classes of points. The classification boundary thus becomes an approximation of the separating surface.

The basic idea behind SVMs is to obtain a function or "classifier" of the form

$$f^*_\lambda(x) = \sum_{i=1}^{N} \alpha_i K(x^i, x), \tag{3.8}$$

where α_i are coefficients associated with locations of the data points x^i, λ is a regularization parameter, and K is a Mercer kernel [25]. Evaluation of the kernel yields the dot product between two points in a higher-dimensional feature space in which a linear classification boundary is sought. This feature space is the reproducing kernel Hilbert space (RKHS) $\mathcal{H}_K$ induced by the kernel. In other words, one can define the mapping $\Phi : \mathcal{X} \to \mathcal{H}_K$ and represent the kernel as $K(x, y) = \Phi^T(x)\Phi(x)$. To find the SVM classifier, however, only this inner product is needed. Thus Φ need not be specified explicitly. This is important because dimensionality of the feature space can be quite large—for example, infinity in the case of a Gaussian kernel.

The SVM classifer is a solution to a regularized least squares problem with hinge loss given by

$$f_\lambda^*(x) = \arg\min_{f\in\mathcal{H}_K}\left\{\frac{1}{n}\sum_{i=0}^{n}\max\left(0, 1-y^i f(x^i)\right)+\lambda||f||^2_{\mathcal{H}_K}\right\}, \tag{3.9}$$

where n is the number of training points, x^i are the training points, y^i are the labels of training point i, $f(x^i)$ is the classifier function evaluated at training point i, and λ is a regularization parameter. From this optimization problem we see that the classifier is determined by its sign: $f_\lambda^*(x) > 0$ if $x \in R_1$ and $f_\lambda^*(x) < 0$ if $x \in R_2$, where R_1 and R_2 are the regions, or classes, bounded by the separating surface. While the sign of the classifier indicates the region/class to which any point belongs, its magnitude reflects the distance a point lies from the boundary in the feature space. Points for which $|f_\lambda^*| < 1$ are said to lie within the margin of the classifer, while larger magnitudes of f_λ^* correspond to points increasingly further from the classifier boundary.

Implementation of the SVM involves selecting a kernel. In this work we use a Gaussian kernel $K(x, y) = \exp\left\{-\|x-y\|^2/2\sigma^2\right\}$. The computational cost of finding the classifier using the SMO algorithm in [30] is problem dependent but can range from $\mathcal{O}(N)$ to $\mathcal{O}(N^2)$ [30]. Additional costs may be incurred depending on the choice of cross validation techniques to select the parameters involved in the kernel (e.g., σ) and the penalty on misclassified training samples, λ. Choosing a small σ or a small λ can lead to large generalization errors because of overfitting, but choosing large values can cause a loss of complexity of the representation (underfitting). For the algorithm described in this work, LIBSVM [6] is used to implement SVMs.

3.3. Uncertainty sampling and active learning. Active learning [8, 34, 36] and specifically US [23] are unsupervised learning techniques commonly used in the machine learning community when labeling data points according to their class is an expensive process. In this context one would like to select, from a large unlabeled set, a small subset of points most useful for constructing or refining a classifier; only the selected points are then labeled. In the discontinuity detection problem, we are free to evaluate the model anywhere in the domain; however, each evaluation is expensive and requires careful selection. US involves only evaluating the model in locations where the classifier is relatively uncertain about the class to which a data point belongs. In these situations US is used to add data points adaptively to a data set, retraining the classifier after each addition.

In the context of SVMs, one may define the uncertainty as the closeness of the evaluating point to the boundary. As described above, this closeness is measured by the magnitude of the classifier function (3.8). An application of SVMs in this context can be found in the reliability design and optimization literature [3], where active learning was used to help refine the boundary of a failure region.

4. Discontinuity detection algorithm. The algorithm presented in this section takes advantage of any regularity exhibited by the separating surface, avoids the creation of structured grids and nested rectangular subdomains, and incorporates guided random sampling to improve scaling with dimension. These features of the algorithm result from an integration of the tools discussed in section 3. PA is used to obtain general information about the the size and location of the discontinuity, and the regularity of the separating surface is exploited by the SVM classifier. Approximating the separating surface using SVMs allows for a more efficient description of the discontinuity than the nearest neighbor approach used in edge tracking and

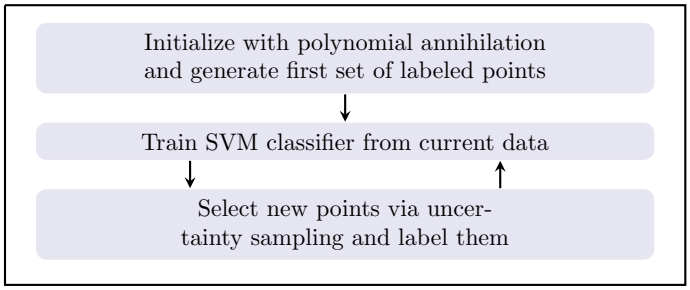


FIG. 1. *Flow chart of the discontinuity detection algorithm.*

adaptive refinement schemes. Additionally, SVMs are robust and tend to not overfit the data due to the regularization described above.

The methodology employed to detect and parameterize the separating surface can be described in three steps, depicted in Figure 1. The first step is an initialization that involves identifying essential characteristics of the discontinuity, such as the jump size and the approximate location of the separating surface, at several points in the parameter domain. This step also seeds the labeling mechanism by which new model evaluations may be classified according to their value. The second and third steps are then alternated repeatedly. The second step involves constructing an SVM classifier to describe the separating surface. The third step involves refining the SVM classifier by selecting new model evaluation points via US and labeling these points.

In the remainder of this section, we will use $x \in \mathbb{R}^d$ to denote the d-dimensional model parameters. The model is now $f : \mathbb{R}^d \to \mathbb{R}$. Subscripted variables denote position along the coordinate axes, i.e., $x_j \in \mathbb{R}$ is the jth coordinate of x, while superscripts are used to index sample points.

4.1. Initialization with polynomial annihilation. The purpose of an initialization phase of the discontinuity detection algorithm is ultimately to provide a mechanism for labeling future model evaluations according to their values. This labeling is necessary to provide a labeled set of points with which to build the SVM classifiers. Note that PA is used to label points according to their *function value*, whereas the SVM is used to label points based upon their *location* in parameter space.

The initialization procedure is essentially a divide-and-conquer approach guided by repeated applications of one-dimensional PA. It is similar to the procedure found in [18]. The procedure begins with an initial set of function evaluations and ends with a set of jump function values at various points in the parameter space. These points are surrounded by additional points at which the model (but not the jump function) was evaluated. One major difference between our implementation and that of [18] involves the selection of points used in each PA calculation. In particular, we define an off-axis tolerance `tol` which is used to define the axial point set $\mathcal{S}(x)$ described in section 3.1. Intuitively, the off-axis tolerance reflects an accepted minimum resolution level of the discontinuity, as described below.

Figure 2 illustrates the application of PA along the horizontal dashed line (the x_j axis), and in particular, our method for choosing the point set $\mathcal{S}_j(x)$ used to perform PA in the jth coordinate direction around a point x. The vertical dashed line denotes all other coordinate directions, $x_{\sim j} \in \mathbb{R}^{d-1}$. The point of interest (POI) x^p, denoted by the pink circle, is the point at which the jump function will be evaluated. The

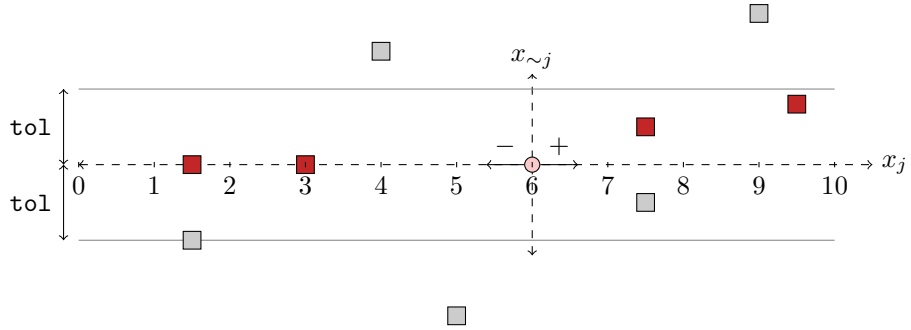


FIG. 2. *Selection of the set $\mathcal{S}_j(x)$ for performing PA along the horizontal axis x_j.*

arrows labeled + and − refer to the relative directions, along the x_j axis, of the surrounding points. For the purposes of PA, at least one point on either side of the POI is necessary. The boxes denote points at which we have performed function evaluations. Two light grey lines bound the region within `tol` of the axis, wherein all points are considered to be "semiaxial" and thus suitable for detecting a discontinuity along x_j. In other words, any of the boxes within the grey lines may be considered for the set $\mathcal{S}_j(x^p)$; those actually selected for this set are drawn in red.

Two special cases are illustrated in Figure 2. The first special case involves the points located at $x_j = 1.5$. These points are equidistant from the POI along the x_j axis, and in this situation the point with the smaller Euclidean distance (in all d directions) from the POI is chosen. The second special case involves the points at $x_j = 7.5$. These points are equidistant from the POI both in the x_j direction and in total distance. In this situation either of the points may be chosen; the top point is chosen here for illustration, but the tie is broken randomly in practice. Once the points available for the stencil are determined, the number actually used for PA is determined by the desired annihilation order m. Following this selection, we approximate the jump function at the POI using (3.2).

The inclusion of semiaxial points in $\mathcal{S}_j(x^p)$ may affect the accuracy of the jump function approximation. We can analyze this effect by considering the error induced by incorrectly evaluating the function values $f(x^l)$ in (3.2). Suppose that we are trying to approximate the jump function at a point x in direction j using semiaxial neighbors $\hat{x}^l \in \mathcal{S}_j(x)$ and $\texttt{tol} > 0$. We perform the PA procedure as if we have truly on-axis points given by $x^l := (x_1, \ldots, x_{j-1}, \hat{x}^l_j, x_{j+1}, \ldots x_d)$, one for each element of $\mathcal{S}_j(x)$. The jump function approximation is computed as

$$\widetilde{L}_m f(x) = \frac{1}{q_m(x)} \sum_{\hat{x}^l \in \mathcal{S}_j(x)} c_l(x) f(\hat{x}^l), \tag{4.1}$$

where we recall that c_l and q_m depend only on the jth coordinate of the points in $\mathcal{S}_j(x)$, and hence their values are equivalent for x^l and $\hat{x}^l$. The difference between the approximation above (4.1) and that in (3.2) is then due only to evaluating f at $\hat{x}^l$ rather than at x^l:

$$\widetilde{L}_m f(x) - L_m f(x) = \frac{1}{q_m(x)} \sum_{\hat{x}^l \in \mathcal{S}_j(x)} c_l(x) \left[f(\hat{x}^l) - f(x^l) \right]. \tag{4.2}$$

If a discontinuity exists in between $\hat{x}^l$ and x^l, then errors introduced into the ap-

proximation will be on the order of the size of the jump. However, if for every $\hat{x}^l$, f is continuous on the closed interval between $\hat{x}^l$ and x^l and differentiable on the corresponding open interval, then we can use the mean value theorem to bound the magnitude of the difference $|f(\hat{x}^l) - f(x^l)|$. Under these conditions, for each l there exists a $\nu^l = (1-\eta^l)x^l + \eta^l \hat{x}^l$ with $\eta^l \in (0,1)$ such that

$$f(\hat{x}^l) - f(x^l) = \sum_{i=1, i\neq j}^{d} \partial_{x_i} f(\nu^l) \left(\hat{x}_i^l - x_i^l\right). \tag{4.3}$$

Then

$$|f(\hat{x}^l) - f(x^l)| \leq \sum_{i=1, i\neq j}^{d} \left|\partial_{x_i} f(\nu^l)\right| \left|\hat{x}_i^l - x_i^l\right| \leq \texttt{tol} \sum_{i=1, i\neq j}^{d} \left|\partial_{x_i} f(\nu^l)\right|. \tag{4.4}$$

Now let $G = \max_l \max_{i\neq j} \sup_{s\in\mathcal{B}_j(x^l)} |\partial_{x_i} f(s)|$, where $\mathcal{B}_j(x^l)$ is a ball of radius $\texttt{tol}$ surrounding x^l in the $x_{\sim j}$ directions, i.e., $\mathcal{B}_j(x^l) := \{(s_1, \ldots, s_{j-1}, x_j^l, s_{j+1}, \ldots, s_d) : |s_i - x_i^l| < \texttt{tol},\ i = 1 \ldots d,\ i \neq j\}$. Then we can bound the difference (4.4) above by $G(d-1)\,\texttt{tol}$. The magnitude of the difference between (3.2) and (4.1) can then be estimated as

$$\left|\widetilde{L}_m f(x) - L_m f(x)\right| \leq \frac{1}{|q_m(x)|} \sum_{\hat{x}^l \in \mathcal{S}_j(x)} |c_l(x)| \left|f(\hat{x}^l) - f(x^l)\right| = \mathcal{O}(Gd\,\texttt{tol}), \tag{4.5}$$

where the second step uses the fact that both $c_l(x)$ and $q_m(x)$ are of the same magnitude, $\mathcal{O}(h(x)^{-m})$ [2]. An application of the triangle inequality then yields an update to the error estimate (3.6) for approximation of the jump function:

$$\widetilde{L}_m f(x) = \begin{cases} [f](\xi) + \mathcal{O}(h(x)) + \mathcal{O}\left(Gd\,\texttt{tol}\right) & \text{if } \hat{x}_j^{l-1} \leq \xi, x_j \leq \hat{x}_j^l, \\ \mathcal{O}(h^{\min(m,k)}(x)) + \mathcal{O}\left(Gd\,\texttt{tol}\right) & \text{if } f \in C^k(I_x) \text{ for } k > 0. \end{cases} \tag{4.6}$$

In this multidimensional case, $\xi \in \mathbb{R}^d$ but differs from x only in dimenson j, i.e., $\xi_i = x_i$ for $i \neq j$, and ξ_j is a location of a jump discontinuity along the x_j axis. I_x and $h(x)$ are defined just as in section 3.1, using only the jth coordinates of the points in the set $\mathcal{S}_j(x)$. This simple estimate suggests that as long as $Gd\,\texttt{tol}$ is significantly smaller than the jump size $[f](\xi)$, only small errors will be induced in the jump function approximation by using off-axis points. Intuitively this means that the off-axis tolerance should be kept small enough to balance the variation of the function in the off-axis directions.

Now that we have described the selection of points used for each one-dimensional application of PA, we describe the multidimensional initialization procedure. This algorithm is based on a repeated divide-and-conquer refinement of some initial set of points. Each refinement further localizes the discontinuity. The core of the divide-and-conquer approach for PA requires the evaluation of the jump function at various test points based on a set of previously evaluated data points. The algorithm is recursive in the sense that at any given step, we wish to refine the location of the discontinuity in direction j and at a given location x. We do this by first finding two additional points at which to evaluate the jump function; these points, y^1 and y^2, are chosen to be the midpoints between x and its nearest semiaxial neighbors in

the $\pm j$ directions. Next, we evaluate the jump function at each of these locations, $J^1 = [f](y^1)$ and $J^2 = [f](y^2)$. If the value of J indicates that a jump exists at either of these points (i.e., up to the accuracy given in (3.6)) then we either evaluate the full model f at the point and perform the same procedure recursively in every other coordinate direction, or we add the point to the set of edge points $\mathcal{E}$ and stop refining around it. Before recursively performing the procedure in a particular direction k for a point y, we evaluate f on the k-semiaxial *boundary parents* corresponding to y if these evaluations do not already exist. These boundary parents are locations on the boundary of the parameter space in the $+k$ and $-k$ directions.[1] Once the function is evaluated at these parent locations, we are assured to have a sufficient number of semiaxial function evaluations to perform PA. The set of edge points $\mathcal{E}$ is the set of points at which we have found nonzero approximations of the jump function and that are located within an *edge tolerance* δ of two other points at which we have evaluated the function. The entire algorithm exits when either no more refinement is possible or the cardinality of the set of edge points reaches a user defined value N_E.

Algorithm 1, `RefinementInitialization`, initializes the refinement of the discontinuity by calling Algorithm 2, `Refine1D`, for each initial point in a set $\mathcal{M}_0$. In practice, we often start either with a single point at the origin or randomly sampled points through out the regime. Initialization with randomly sampled points can provide a more robust method for finding the separating surface since they force an exploration of a wider area of the parameter domain. `Refine1D` recursively refines the location of the discontinuity as described above. Both are detailed below, and they constitute the PA phase of the overall discontinuity detection algorithm. For reference, the function $\mathrm{NN}_{\pm k}(\mathcal{S}, x)$ finds the nearest neighbor to the point x in the $\pm k$ coordinate direction, among the points in the set $\mathcal{S}$.

ALGORITHM 1. `RefinementInitialization`.

Input: initial point set $\mathcal{M}_0$; maximum number of edge points N_E; edge tolerance δ; off-axis tolerance `tol`
Initialize: $\mathcal{M} = \mathcal{M}_0$, $\mathcal{E} = \emptyset$, $\mathcal{F} = \{f(x^1), f(x^2), \ldots, f(x^n) : x^i \in \mathcal{M}\}$
for all $x^i \in \mathcal{M}$ **do**
  for j from 1 to d **do**
    If needed, add boundary parents of x^i in direction j and their function values to $\mathcal{M}$ and $\mathcal{F}$, respectively
    $(\mathcal{M}, \mathcal{E}, \mathcal{F}) =$ `Refine1D`$(\mathcal{M}, \mathcal{E}, \mathcal{F}, x^i, j, N_E, \delta,$ `tol`$)$
    if $|\mathcal{E}| \geq N_E$ **then**
      Return $(\mathcal{M}, \mathcal{E}, \mathcal{F})$
    end if
  end for
end for
Return $(\mathcal{M}, \mathcal{E}, \mathcal{F})$

4.2. Labeling in the initialization phase. Having estimated the jump size and location of the discontinuity at a few points in the parameter space using PA, we would like to use these estimates to *label* the points in $\mathcal{M}$ according to the function evaluations already performed by the initialization procedure (and stored in $\mathcal{F}$).

[1] If the parameter space is unbounded, then the boundary parents can be any k-semiaxial points that are far enough from y in the k direction to ensure that the stencil for jump function evaluation at y is not too narrow.

ALGORITHM 2. `Refine1D`.

Input: point set $\mathcal{M}$; edge point set $\mathcal{E}$; model evaluations $\mathcal{F}$; location for refinement x; coordinate direction of refinement j; maximum number of edge points N_E; edge tolerance δ; off-axis tolerance `tol`
Determine $\mathcal{S} := \mathcal{S}_j(x)$ using tolerance `tol`.
Define $y^1 = (x + \mathrm{NN}_{+j}(S, x))/2$
Define $y^2 = (x + \mathrm{NN}_{-j}(S, x))/2$
$J^1 = [f](y^1)$
$J^2 = [f](y^2)$
for each $k \in 1, 2$ **do**
  if J^k indicates jump exists **then**
    if $\|y^k - x\| \leq \delta$ **then**
      Add y^k to $\mathcal{E}$.
      if $|\mathcal{E}| \geq N_E$ **then**
        Return $(\mathcal{M}, \mathcal{E}, \mathcal{F})$
      end if
    else
      Add y^k to $\mathcal{M}$.
      Add $f(y^k)$ to $\mathcal{F}$.
      for l from 1 to d **do**
        If needed, add boundary parents of y^k in direction l and their function values to $\mathcal{M}$ and $\mathcal{F}$, respectively
        $(\mathcal{M}, \mathcal{E}, \mathcal{F})$ = Refine1D($\mathcal{M}$, $\mathcal{E}$, $\mathcal{F}$, y^k, l, N_E, δ, `tol`)
      end for
    end if
  end if
end for
Return $(\mathcal{M}, \mathcal{E}, \mathcal{F})$

Determining the class in which a point resides depends on the jump values at the edge points. We only label the points in $\mathcal{M}$ that lie within the edge tolerance δ of points in $\mathcal{E}$. Recall that each edge point (i.e., each element of $\mathcal{E}$) lies within δ of at least two points in $\mathcal{M}$.

For each point $y \in \mathcal{E}$, we find the elements of $\mathcal{M}$ within δ of y; call these $\mathcal{M}_y = \{x : x \in \mathcal{M}, |x - y| < \delta\}$. Of this subset, the point x^* with the largest function value is found and labeled *class* 1. Then the function values at all the other points $x \in \mathcal{M}_y$ are compared to $f(x^*)$ using the jump value $[f](y)$ as a reference. If the difference between $f(x)$ and $f(x^*)$ is less than the jump value, then the point x is labeled *class* 1; otherwise, it is labeled *class* 2. We note that *class* 1 therefore always contains the locally largest values by definition. (If it is known a priori that the locally largest values in different parts of the domain should be in different classes, a different labeling procedure that incorporates this knowledge must be used.) The present procedure can successfully label points along a discontinuity whose jump size varies along the domain, without any manual intervention. Figure 3 illustrates the procedure.

The complete initialization phase of the algorithm is now demonstrated on three test discontinuities, shown in Figure 4. In these numerical experiments, δ and `tol` are both set at 0.125, and $\mathcal{M}_0$ consists of a single point at the origin. In these plots we see that the discontinuity is located and surrounded by a very coarse grid of function

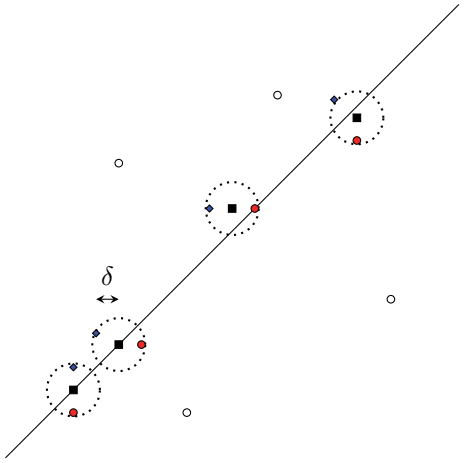


FIG. 3. *PA-based labeling procedure, used during the initialization phase of the algorithm. Circles and diamonds are locations where the function has been evaluated. Squares are edge points. Blue diamonds are function evaluations that are labeled as class* 1*, and red circles are function evaluations that are labeled as class* 2*. The dotted circles are of radius* δ*.*

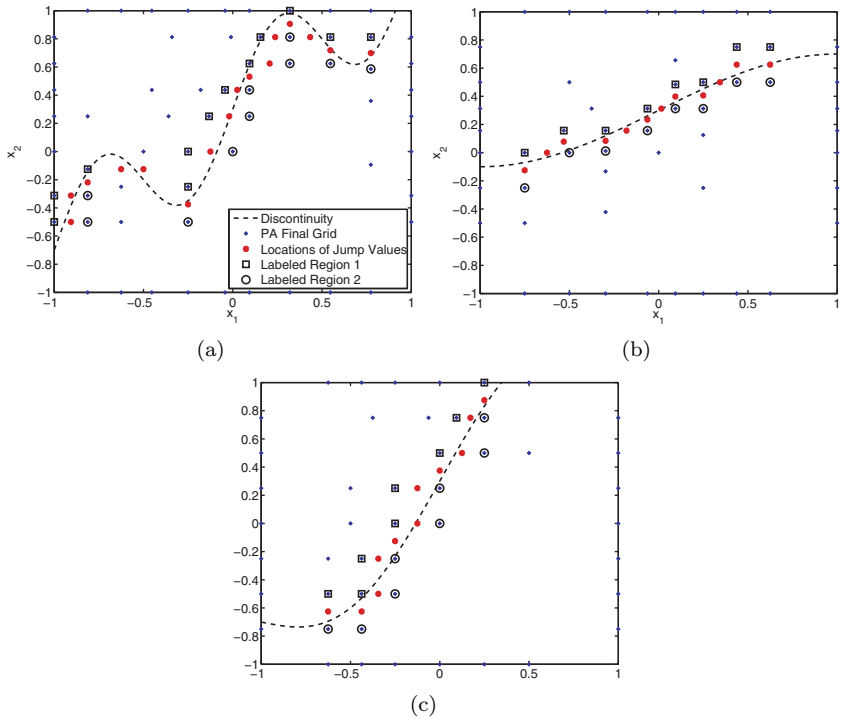


FIG. 4. *Initialization phase of the discontinuity detection approach (Algorithm* 1*), applied to several discontinuities. All model evaluation points are marked with diamonds; labeled points have an additional circle or square. Edge points are indicated with red circles.*

evaluations. The red circles points indicate locations at which we obtain jump values approximating the size of the discontinuity. These are the points in set $\mathcal{E}$, used to label the surrounding function evaluations as described above.

4.3. Refinement with uncertainty sampling. We now describe the use of SVM classification and active learning to refine our description of the discontinuity, following the initialization phase of the algorithm, i.e., Algorithm 1. Compared to simply continuing Algorithm 1 with smaller tolerances to generate more edge points, this phase of the algorithm focuses on choosing model evaluation points that are most informative for the SVM classifier. Later we will demonstrate, via numerical examples in section 5.3, that switching to the active learning phase after relatively few iterations of Algorithm 1 results in significant efficiency gains and improved scaling of computational effort with parameter dimension. The steps described in this section comprise the second and third boxes in the flow chart of Figure 1: SVM classification, using all of the currently labeled points (*class* 1, *class* 2), alternates with the selection of new points via US and the labeling of these new points.

New points are chosen based on their proximity to the zero level set of the current classifier. We obtain a point near the classifier boundary by first *drawing a sample* from an underlying measure on x (e.g., a probability measure on the input parameters of the model) and then using this sample as an initial guess for the following optimization problem, which minimizes the square of the classifier function in (3.8) and thus drives the initial guess towards the boundary:

$$\min_x \left(\sum_{i=1}^{N} \alpha_i K(x^i, x) \right)^2 . \tag{4.7}$$

Here, $\{x^1, \ldots, x^N\}$ are the support vectors of the current classifier. A variety of optimization algorithms can be used for this purpose and result in similar performance. Note that this optimization problem is not convex and may have multiple local minima. But the initial randomization helps mitigate clustering of points in local minima, and in practice the possibility of clustering does not impede refinement of the discontinuity; as the SVM is updated, these minima are themselves altered. Moreover, we compel the generation of "low discrepancy" points along the discontinuity by constraining new function evaluations to occur farther than a minimum distance ϵ from existing points. In other words, a candidate point x^* found by minimizing 4.7 is not evaluated or used for classification if it lies less than ϵ away from the nearest evaluated node. As US progresses, the entire discontinuity will be explored with resolution ϵ. Eventually the algorithm results in randomly distributed training points that approximate a Monte Carlo edge tracking scheme. The US scheme is precisely detailed in Algorithm 3.

When a new data point is generated through US, it must be assigned to a class. Our labeling scheme for the points generated during PA relied on estimates of the jump function and thus only applied to points within δ of an edge point. Now during US, we must label points that potentially lie much further away from edge points, where no local value of the jump function is available. We thus employ a different labeling scheme that compares new points generated during uncertainty sampling to the nearest previously labeled points in each class.

In particular we define a new tolerance δ_t which reflects the radius of a region in each class within which the *local* variability of the function along the separating surface is smaller than the local jump size itself. In principle we can specify a separate δ_t for each class, but for simplicity we consider the same value for both. A new point is now labeled only if its nearest neighbors in each class are located within a distance δ_t. The function value at the new point is compared to the function value of its nearest neighbor in *class* 1 and its nearest neighbor in *class* 2. The new point is given the

ALGORITHM 3. `FindPointsOnClassifierBoundary`.

Input: set $\mathcal{L} = \{(x, \ell)\}$ of (points, labels); number of points to add N_{add}; variation radius δ_t; resolution level $\epsilon < \delta_t$; maximum number of iterations `itermax`

$N_{added} = 0$
$\mathcal{X} = \emptyset$
`iter` $= 1$
while $N_{added} < N_{add}$ and `iter` $<$ `itermax` **do**
  $x \leftarrow$ `samplePointFromDomain()` {equation (4.7)}
  if $\arg\min_{(x^*,\ell^*)\in\mathcal{L}} ||x - x^*|| > \epsilon$ **then**
    if $\exists(x^1, \ell^1) \in \mathcal{L}$ s.t. $||x^1 - x|| < \delta_t$ and $\ell^1 = +1$ **then**
      if $\exists(x^2, \ell^2) \in \mathcal{L}$ s.t. $||x^2 - x|| < \delta_t$ and $\ell^2 = -1$ **then**
        $N_{added} = N_{added} + 1$
        $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$
      end if
    end if
  end if
  `iter` $=$ `iter` $+ 1$
  Return($\mathcal{X}$)
end while

same label as the nearest neighbor with the closest function value. For this scheme to avoid making any errors, δ_t must be chosen properly. We explain this requirement and precisely define the notion of "local" as follows. Suppose that we are attempting to label a new point x^u which has function value $f(x^u)$ and that its nearest neighbors in *class* 1 and *class* 2 are $x^{(1)}$ and $x^{(2)}$, respectively. Suppose also that both nearest neighbors are within δ_t of x^u. Based on the class definitions in section 4.2, we can assume that $f(x^{(1)}) > f(x^{(2)})$. We now determine the consequences of our labeling mechanism if x^u lies in *class* 1. There are three possible orderings of $f(x^u)$ relative to $f(x^{(1)})$ and $f(x^{(2)})$. If $f(x^u) > f(x^{(1)}) > f(x^{(2)})$, then our scheme will generate the correct label, because the function value of the nearest *class* 1 point is closer to that of the new point. If $f(x^u) < f(x^{(2)}) < f(x^{(1)})$, then we will generate an incorrect label; in this situation, the variation of the function *within class* 1*, near the discontinuity* exceeds the jump size $|f(x^{(1)}) - f(x^{(2)})|$. The points $x^{(1)}$ and $x^{(2)}$ are too far from x^u to be useful for labeling, and thus δ_t has been chosen too large. The final possible ordering is $f(x^{(2)}) < f(x^u) < f(x^{(1)})$. In this case, we can still label the point correctly if $f(x^{(1)}) - f(x^u) < \frac{1}{2}|f(x^{(1)}) - f(x^{(2)})|$. Alternatively, if x^u belonged to *class* 2, we would need $f(x^u) - f(x^{(2)}) < \frac{1}{2}|f(x^{(1)}) - f(x^{(2)})|$. To ensure that the appropriate inequalities hold, the radius δ_t must be specified so that the variation of the function around the new point in each class is smaller than $\frac{1}{2}|f(x^{(1)}) - f(x^{(2)})|$. This radius reflects a region *within* a given class in which the function varies a small amount relative to the jump size. If the radius is any larger, this labeling procedure may not be accurate for this final ordering. If the jump size is large relative to the local variation of the function throughout the parameter domain, then δ_t can be quite large, even infinity. If the function values near the separating surface within a particular class vary widely, however, then a smaller δ_t is needed to ensure accurate labeling. In order to conservatively choose δ_t, one may set $\delta_t = \delta$, i.e., the same as the edge tolerance. In this situation, US begins labeling new points which are near existing labeled PA points. These labels will be correct because these points are in a region where we have an accurate approximation of the jump size. Alternatively, one may

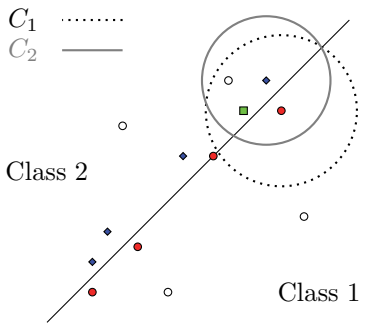


FIG. 5. *Labeling procedure for points generated during US. A new test point (green square) is labeled by comparing its function value to those of its nearest neighbors in Class* 1 *and Class* 2*, but only if the test point is sufficiently close to both neighbors. The blue diamonds and red circles denote previously labeled points from Class* 1 *and* 2*, respectively. Circles* C_1 *and* C_2 *indicate the regions for each class in which the local variability of the function is small enough for labeling. The radii of these circles are* δ_t^1 *and* δ_t^2 *and are chosen based on the discussion in section* 4.3*. New points within the intersection of these circles can be accurately labeled.*

choose to perform PA with a larger total number of edge points N_E or a smaller edge tolerance δ. These changes would yield a more extensive exploration of the separating surface in the PA initialization phase of the algorithm, obtaining jump value estimates at more areas along the separating surface. The main consequence of setting δ_t too small or performing a large amount of PA is a loss in efficiency; only samples very close to existing samples will be labeled, and progress along the discontinuity will be slow. But the conditions favoring large δ_t are likely to be valid in practice, as many discontinuities in problems of interest involve relatively large jumps. An example of this labeling procedure is shown in Figure 5, where the radius δ_t is different in each class for illustration purposes. In practice we specify δ_t to be equal in each class.

Once the new point is labeled, a new classifier is trained and the procedure is repeated: US, labeling, and SVM training. If a sufficient number of function evaluations are added, this procedure will revert to a Monte Carlo edge tracking scheme with resolution level ϵ, with the SVM interpolating the separating surface among the support vectors. US can also be used to add several new points at a time, by minimizing (4.7) from several starting points at each iteration. This does not typically reduce the number of function evaluations needed to refine the discontinuity but can lower overall computational cost by reducing the number of classifiers to be trained.

4.4. Stopping criterion. The stopping criterion for the overall discontinuity detection algorithm is specified by the distance tolerance ϵ described above (i.e., the distance used to reject too-closely spaced points during US). For a given value of ϵ, eventually a sufficient number of points are added so that any additional point along the discontinuity lies within ϵ of a previously labeled point;[2] at this stage the algorithm exits. In practice, the exit criterion is implemented by making many repeated attempts at adding new samples and exiting after a specified number of failed attempts occur in sequence. In Algorithm 3, this number of attempts is given by `itermax`.

We also considered using cross validation as a stopping criterion but found it to be inadequate for this purpose. The reason is that cross validation can only indicate whether points that are *already labeled* are correctly classified by the SVM. These

[2]In the case of an unbounded parameter domain endowed with finite probability measure, this will only hold true with high probability.

points are not distributed randomly over the entire domain; rather, they are clustered around areas of discontinuity that have already been identified. Cross validation thus has no means of revealing whether the entire discontinuity has been explored. The only way to be sure that the entire discontinuity has been explored is to add points until a desired resolution level is achieved along the entire discontinuity, namely, ϵ.

Using this stopping criterion and running the algorithm to completion, the number of points required will grow exponentially with the dimension of the separating surface. Even though the algorithm involves random sampling, the stopping criterion essentially expresses the desire to achieve a space-filling design along the separating surface. In practice, however, we have found that stopping well short of a small ϵ still leads to good results. This behavior can be attributed to random sampling along the separating surface. Random sampling allows wide regions of the discontinuity to be explored asynchronously, with classification errors resulting from gaps between the samples. Because the exploration is asynchronous and spatially distributed, it is easier to stop the algorithm anytime. These exploration contrasts with edge tracking, where one progressively adds samples by "walking" along the separating surface; here, one cannot stop short because large regions of the separating surface have not yet been explored.

The full discontinuity detection algorithm is summarized in Algorithm 4. It takes as inputs an initial point set $\mathcal{M}_0$, the desired number of edge points N_E, a PA edge tolerance δ, a PA off-axis tolerance `tol`, a function variation tolerance δ_t for US labeling, the number of points to add with every iteration of US N_{add}, an US resolution level ϵ, the maximum number of US subiterations `itermax`, and finally a maximum run time T. The algorithm returns the classifier function f^*_λ.

5. Numerical examples. We now demonstrate the performance of the new discontinuity detection algorithm on a variety of problems: separating surfaces of varying complexity, a problem where the jump size varies along the discontinuity, and discontinuities of increasing dimension. Then we apply the algorithm to an ODE

ALGORITHM 4. `Discontinuity Detection`.

Input: initial point set $\mathcal{M}_0 = \{x^1, x^2, \ldots, x^n | x^i \in \mathbb{R}^d\}$; maximum number of edge points N_E; PA edge tolerance δ; PA off-axis tolerance `tol`; US variation radius δ_t; number of US points added each iteration N_{add}; US resolution level ϵ; maximum number of US subiterations `itermax`; maximum run time T
$\mathcal{M}, \mathcal{E}, \mathcal{F} =$ `RefinementInitialization`$(\mathcal{M}_0, N_E, \delta,$ `tol`$)$
$\{(x, \ell)\} =$ `generateLabels`$(\mathcal{M}, \mathcal{E}, \mathcal{F})$ {Figure 3}
$f^*_\lambda(x) =$ `trainSVMClassifier`$(\{(x, \ell)\})$
while `Runtime` $< T$ **do**
  $\{x^{new}\} =$ `findPointsOnClassifierBoundary`$(\{(x, \ell)\}, N_{add}, \delta_t, \epsilon,$ `itermax`$)$
  if $\{x^{new}\} = \emptyset$ **then**
    Return(f^*_λ)
  end if
  $\{y^{new}\} = f(\{x^{new}\})$
  $\{\ell^{new}\} =$ `label`$(\{x^{new}\}, \{y^{new}\})$ {Figure 5}
  $\{(x, \ell)\} \to \{(x, \ell)\} \cup \{(x^{new}, \ell^{new})\}$
  $f^*_\lambda =$ `trainSVMClassifier`$(\{(x, \ell)\})$
end while
Return(f^*_λ)

system whose fixed point depends discontinuously on its parameters, and finally we evaluate the performance of the algorithm on a problem where a discontinuity exists in a subspace of the full parameter domain.

5.1. Geometry of the separating surface. To evaluate how the performance of the algorithm depends on the regularity of the separating surface, we consider four increasingly complex discontinuity geometries, all in the two-dimensional parameter space $D = [-1, 1]^2$. The first three separating surfaces are given by (5.1)–(5.3) and illustrated in Figures 6–8. The final separating surface is a combination of (5.2) and a rectangle and serves as an example of a discontinuity bounding regions that are not simply connected; this surface is illustrated in Figure 9.

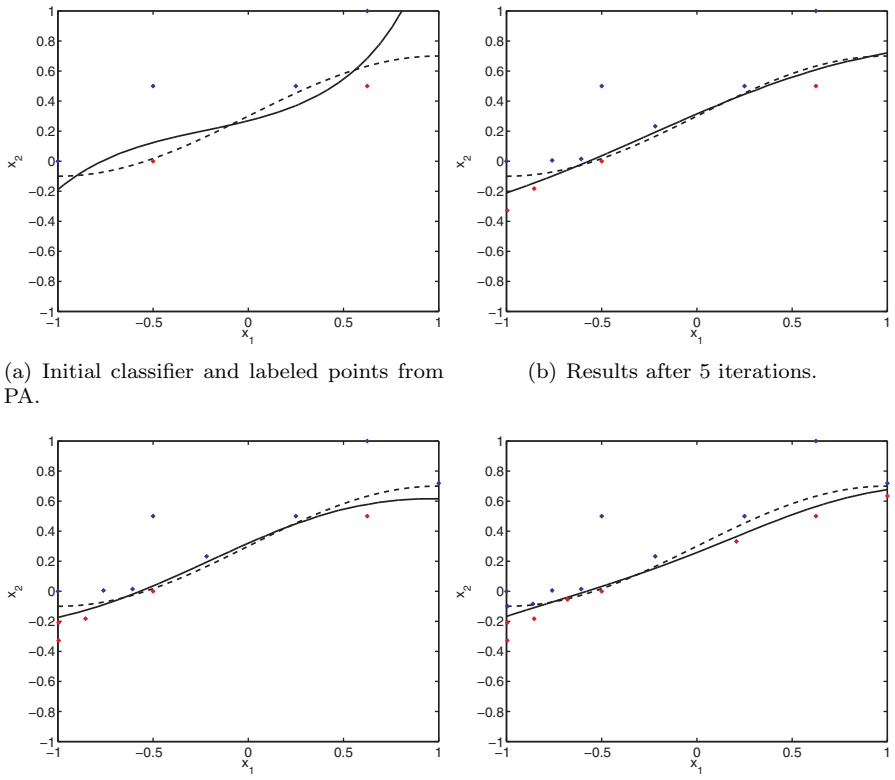
$$x_2 = 0.3 + 0.4\sin(\pi x_1) \tag{5.1}$$

$$x_2 = 0.3 + 0.4\sin(\pi x_1) + x_1 \tag{5.2}$$

$$x_2 = 0.3 + 0.4\sin(2\pi x_1) + x_1. \tag{5.3}$$

Because this example is intended to focus on the geometry of the separating surface, the function in the two regions simply takes values of $+1$ and -1.

The initialization phase of the algorithm is performed with an equal off-axis and edge tolerance $\texttt{tol} = \delta = 0.5$. US is performed with $\delta_t = 2$ and $\epsilon = 0.01$, indi-



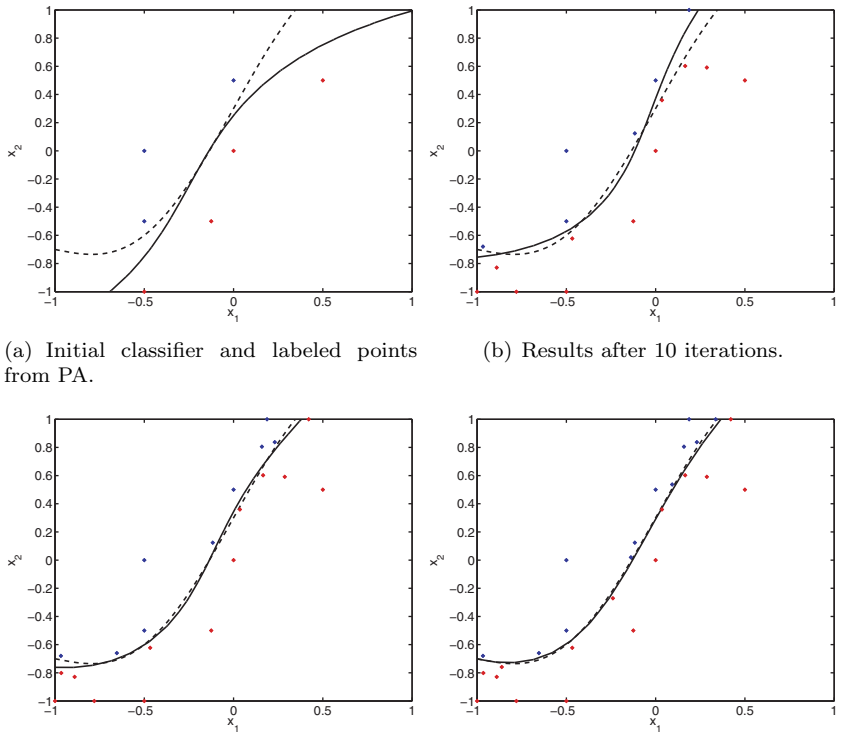
(a) Initial classifier and labeled points from PA.

(b) Results after 5 iterations.

(c) Results after 10 iterations.

(d) Results after 15 iterations.

FIG. 6. *US results for separating surface given in equation* (5.1). *Red and blue points are training samples from each class. The dotted line represents the true separating surface, and the solid black line represents the zero level set of the SVM.*



(a) Initial classifier and labeled points from PA.

(b) Results after 10 iterations.

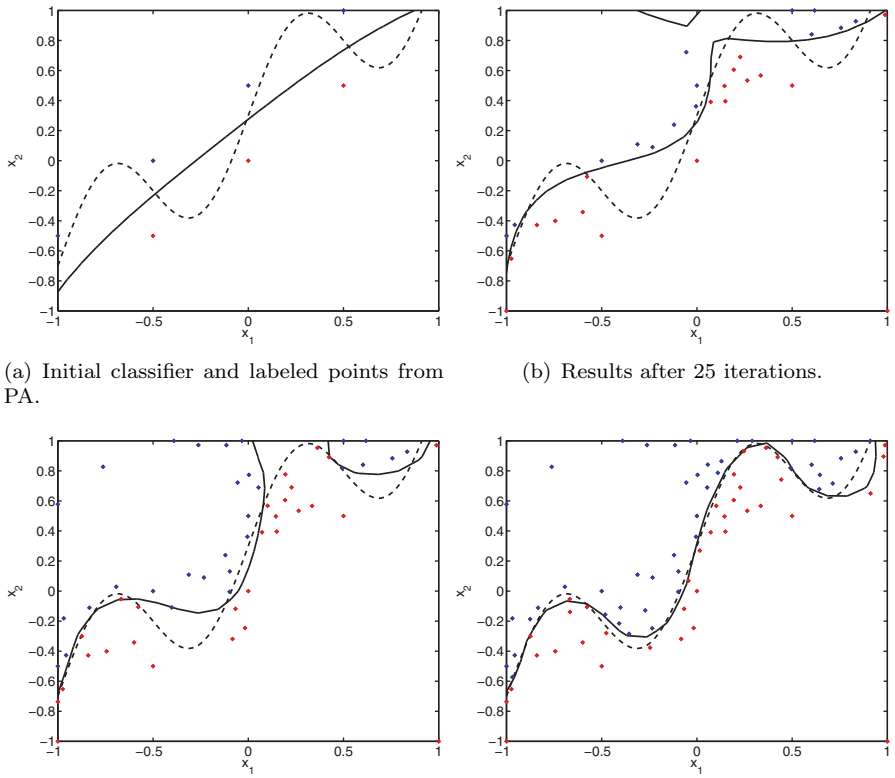
(c) Results after 15 iterations.

(d) Results after 20 iterations.

FIG. 7. *US results for separating surface given in equation* (5.2). *Red and blue points are training samples from each class. The dotted line represents the true separating surface, the solid black line represents the zero level set of the SVM.*

cating that we believe the variation of the function is small compared to the jump size and allows newly sampled points to be fairly close together. We note that in these scenarios the function is constant (exhibits no variation) within each class. The SVM classifier is trained using Gaussian kernels, with parameters chosen via cross-validation. Figures 6–9 show how the distributions of positively/negatively labeled points and the classifier boundary evolve after various iterations of uncertain sampling. The first discontinuity (5.1) is almost linear and requires the smallest number of function evaluations to be accurately captured. The second discontinuity (5.2) is fairly linear over a large region but contains a tail near the lower left-hand corner. The refinement phase of the algorithm effectively locates this tail and accurately creates an approximation of the separating surface. The third discontinuity (5.3) has an oscillatory separating surface and requires the largest number of function evaluations in order to create an accurate classifier. Results for the fourth discontinuity show that the US/SVM approach is capable of identifying and refining separating surfaces that are disjoint.

A quantitative assessment of classifier accuracy and convergence is given in Figure 10. To describe the classifier accuracy, we consider 10000 points sampled from a uniform distribution on the parameter domain and evaluate the percentage of these points that are classified incorrectly. For each of the four discontinuities, we plot the fraction of misclassified points versus the number of model evaluations. Since the discontinuity detection algorithm involves random sampling, we actually run it 100



(a) Initial classifier and labeled points from PA.

(b) Results after 25 iterations.

(c) Results after after 50 iterations.

(d) Results after 75 iterations.

FIG. 8. *US results for separating surface given in equation* (5.3). *Red and blue points are training samples from each class. The dotted line represents the true separating surface, the solid black line represents the zero level set of the SVM.*

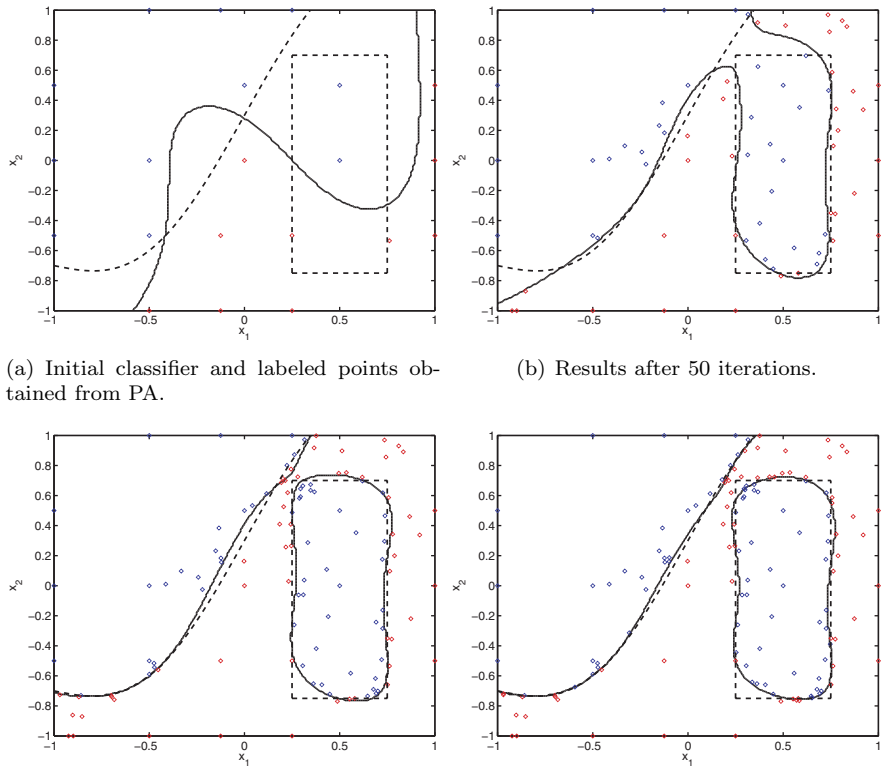
times for each case and plot the mean and standard deviation of the misclassification percentage. Clearly, the more complex discontinuity geometries require more points to achieve an accurate approximation. But errors below 1% are achieved for all four cases.

5.2. Variable jump size. Now we demonstrate the performance of our algorithm on a discontinuity whose jump size is not constant along the separating surface. An example proposed in [7] of such a discontinuity is given in Figure 11. This function arises from the solution of Burgers' equation with a parameterized initial condition. In particular, we have

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) = \frac{\partial}{\partial x}\left(\frac{\sin^2 x}{2}\right), \quad 0 \le x \le \pi,\ t > 0, \tag{5.4}$$

with initial condition $u(x,0) = y \sin x$, $y \sim \mathcal{U}(0,1)$, and boundary conditions $u(0,t) = u(\pi,t) = 0$. The surface in Figure 11 is the steady-state solution of (5.4) plotted as a function of y, i.e., it is $\bar{u}(x,y) := u(x, t = \infty; y)$.

In this experiment we set the labeling radius δ_t for US points to 0.5. A rationale for this choice is as follows. First, note that the minimum jump size in Figure 11, occurring near the $(x,y) = (1,1)$ corner, is approximately 0.5, with function values varying from -0.25 to 0.25. Moving 0.5 units away from this corner along the



(a) Initial classifier and labeled points obtained from PA.

(b) Results after 50 iterations.

(c) Results after after 100 iterations.

(d) Results after 120 iterations.

FIG. 9. *US results for separating surface given* (5.2) *with the addition of a box. Red and blue points are training samples from each class. The dotted line represents the true separating surface, and the solid black line represents the zero level set of the SVM.*

discontinuity, we find that function values near the discontinuity are approximately $f_{\min} = -0.7$ and $f_{\max} = 0.7$, resulting in a jump size of $[f] = 1.4$ and a half-jump size of $[f]/2 = 0.7$. Now suppose that one has already labeled some training points in this region and would like to label new US points in the $(1, 1)$ corner. If $f(x, y) \approx 0.25$ for some point (x, y) in this corner, then (x, y) is in class 1 and $f_{\max} - f(x, y) < [f]/2$; hence our labeling radius is valid for class 1. Now consider the other class in the same corner. In the worst case situation, we will obtain a function value of $f(x, y) \approx -0.25$. Now $f(x, y) - f_{\min} < [f]/2$, and therefore points from class 2 can be labeled as well.

For this value of δ_t, we now explore the impact of varying the edge tolerance δ in the PA phase of the algorithm. We consider $\delta \in \{1/8, 1/16, 1/32\}$ and show convergence results in Figure 12. For each of these refinement levels, we achieve a 1% classification error on 5000 samples after approximately six US iterations. Increasing the refinement of the PA phase of the algorithm increases the number of labeled samples initially fed to the SVM classifier, but this additional work does not seem to be useful. The active learning procedure correctly learns the discontinuity even when the initial PA grid is quite coarse.

5.3. Dimension scaling. Now we evaluate the performance of the discontinuity detection algorithm on higher-dimensional problems, examining the dimension scaling of the initialization (Algorithm 1) and US phases of the algorithm. Consider a function

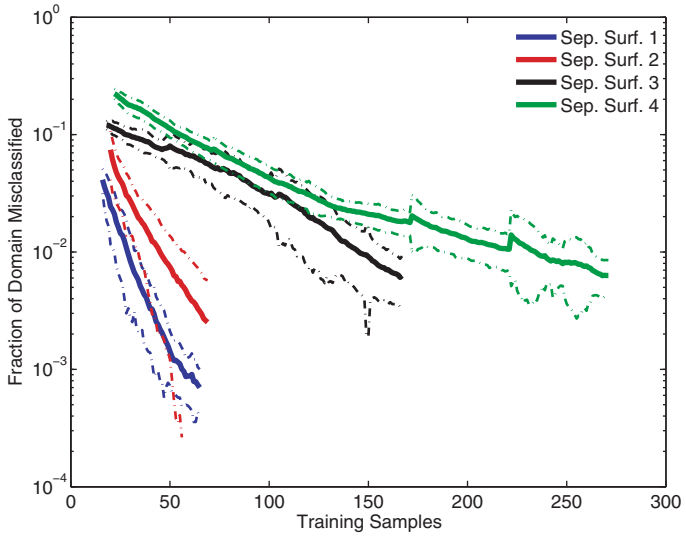


FIG. 10. *Classifier convergence for different discontinuity geometries, specified in section* 5.1. *The fraction of the parameter domain that is misclassified is plotted versus the number of model evaluations. The discontinuity detection algorithm is run* 100 *times; dotted lines indicate errors* $\pm$ *one standard deviation away from the mean. The classifier for each separating surface achieves an error of less than* 1%.

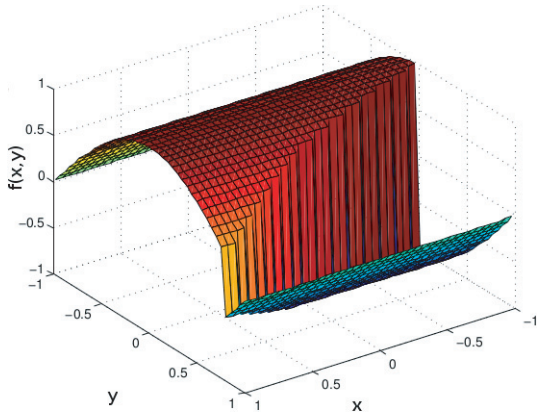


FIG. 11. *Steady state solution of Burgers' equation plotted as a function of the spatial coordinate* x *and the initial condition parameter* y. *The jump size varies along the discontinuity.*

$f : [-1,1]^d \to \mathbb{R}$ with $x := (x_1, \ldots, x_d) \in \mathbb{R}^d$:

$$f(x) = \begin{cases} x^2 + 10 & \text{if } x_d > \sum_{i=1}^{d-1} x_i^3, \\ x^2 - 10 & \text{otherwise.} \end{cases} \tag{5.5}$$

This function is piecewise quadratic with a cubic separating surface. The SVM again employs a Gaussian kernel, and US adds $N_{add} = 10$ points at a time. We vary the parameter dimension d from 2 to 10 and use 10000 points uniformly sampled on the domain of f to evaluate the accuracy of discontinuity localization. For each value of

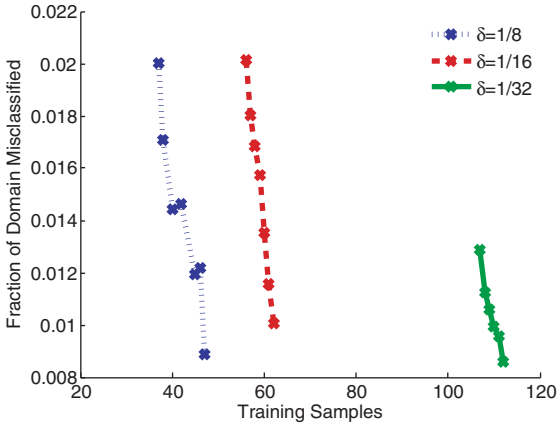


FIG. 12. *Burgers' equation example: convergence of the discontinuity detection algorithm after initialization with different edge tolerances* δ.

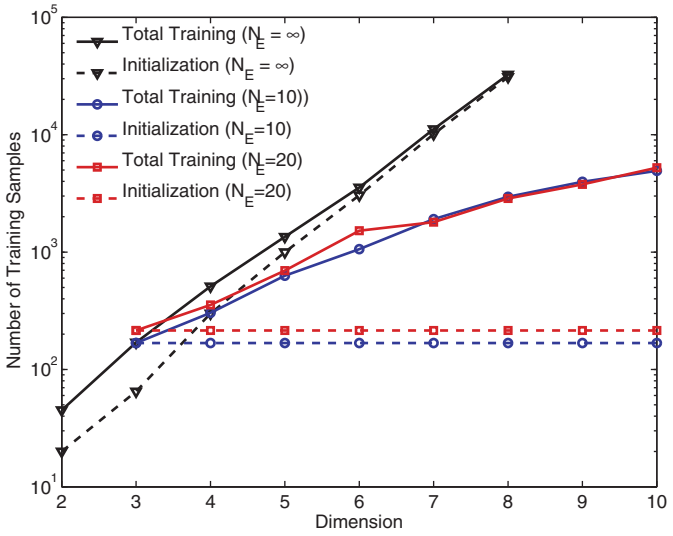


FIG. 13. *Dimension scaling of different refinement algorithms for the cubic separating surface in section* 5.3. *All algorithms are run until* 99% *of the domain is correctly classified. Solid lines indicate the total number of function evaluations, while dashed lines indicate the number of function evaluations performed in the initialization phase of the algorithm only. Three cases are shown, corresponding to different limits* N_E *on the number of edge points produced during initialization.*

d, we run the algorithm until 99% of these points are classified correctly and plot the total number of function evaluations thus invoked. Results are shown in Figure 13.

Figure 13 actually shows the results of three experiments, varying the computational effort devoted to the initialization phase of the algorithm. In the first experiment, PA is performed with an edge tolerance $\delta = 0.25$ and $N_E = \infty$; in other words, we run Algorithm 1 until no more refinement is possible, according to the edge tolerance. This is referred to as performing PA "to completion." In the second and third experiments, we set the number of edge points N_E to 20 and 10, respectively. In all three cases, we then proceed with the SVM classification/US phase of the algorithm until 99% accuracy is achieved. Solid lines show the total number of function

evaluations in each case, while dashed lines show the number of function evaluations performed in the initialization phase only.

The results demonstrate, first, that the number of function evaluations required to perform PA to completion grows exponentially with dimension. This effort completely dominates the total computational cost of the algorithm for $d > 3$. But these results also show that performing PA to completion is not necessary. Stopping at $N_E = 10$ or $N_E = 20$ still allows an accurate characterization of the separating surface to be constructed via US and the associated labeling. In fact, increasing the number of edge points from 10 to 20 does not really affect the number of training samples subsequently required to achieve 99% accuracy. One reason this may be the case is that the discontinuity size is fairly consistent across the domain, and therefore once its magnitude is known, additional edge points are unhelpful. As d increases, the efficiency of the algorithm with finite N_E appears to be several orders of magnitude improved over the scenario in which we perform PA until no further refinement is possible.

5.4. Genetic toggle switch. A differential-algebraic model of a genetic circuit implemented in E. coli plasmids has been proposed in [13]. This model has been frequently used in the computational science literature as a testbed for uncertainty propagation [40], parameter inference [24, 27], and discontinuity detection [18]. The last two studies are concerned with the fact that the system can exhibit a bifurcation with respect to its parameters. The differential algebraic equation model is as follows:

$$\begin{aligned} \frac{du}{dt} &= \frac{\alpha_1}{1+\nu^\beta} - u\,, \\ \frac{dv}{dt} &= \frac{\alpha_2}{1+w^\gamma} - v\,, \\ w &= \frac{u}{(1+[\mathrm{IPTG}]/K)^\eta}\,. \end{aligned} \tag{5.6}$$

The states u and v essentially represent the expression levels of two different genes. The meanings of the parameters α_1, α_2, β, γ, η, K, and [IPTG] are described in [24] and [13] but are not particularly important here. Following [18] and [1], we fix $[\mathrm{IPTG}] = 4.0 \times 10^{-5}$, $\gamma = 1$, and $\beta = 2.5$. We consider variation in four parameters, $Z := (\alpha_1, \alpha_2, \eta, K)$. We let Z vary uniformly within a hypercube around the nominal value $Z_0 := (156.25, 15.6, 2.0015, 2.9618 \times 10^{-5})$ with a range of $\pm 10\%$ in every direction. The output of interest is the steady-state value $v(t = \infty)$.

We apply the discontinuity detection algorithm to this output, with parameters $\delta = \texttt{tol} = 0.25$, $N_E = 15$, $\delta_t = 2.0$, and $\epsilon = 0.01$. The PA parameter choices δ, `tol`, and N_E are all targeted toward achieving a small number of function evaluations during PA. In this model, it is known that the function values exhibit small variability compared to the jump size. For this reason we only need to learn the function values in each class at a few locations in the parameter domain. Once we obtain this information, we have essentially learned the jump size over the entire parameter domain. We can also use a fairly large δ_t, in this case encompassing a majority of the parameter domain. These choices demonstrate the flexibility of the algorithm, in that the parameter choices can reflect prior knowledge when it is available. Finally, the algorithm is fairly insensitive to ϵ because we will employ a stop-short mechanism for termination.

Table 1 shows the number of model evaluations (i.e., integrations of (5.6)) required to localize the discontinuity to within a 1% classification error and a 0.1% classification

TABLE 1
Performance comparison of different discontinuity localization schemes on the genetic toggle switch example.

	Learning	Edge tracking [18]	Adaptive refinement[1]
Model evals for 1% error	127 ± 2	31,379	91,250
Model evals for 0.1% error	257 ± 22	–	–

error, measured with 5000 random samples from the parameter domain. Since our algorithm involves random sampling, we report the average and standard deviation of this value over 100 independent runs. For comparison, we also report the number of model evaluations used by the edge tracking scheme of [18] and adaptive refinement scheme of [1] for exactly the same problem. The performance of the new discontinuity localization algorithm is much improved over these previous techniques. We attribute the improvement to the fact that the separating surface in this example can be well described using a hyperplane, as in [1]. The SVM classifer needs very few points to approximate a hyperplane or small perturbations thereof.

5.5. Discontinuity in subspace of full domain. Finally, we demonstrate the performance of the discontinuity detection scheme on a problem wherein the discontinuity occurs only along a subset of the input coordinates. In other words, the problem contains a separating surface that is aligned with a complementary subset of the coordinate directions. In particular, we consider a function $f : [-1, 1]^{20} \to \mathbb{R}$ containing a discontinuity across a 2-sphere that is "extruded" through a 20-dimensional ambient space:

$$f(x) = \begin{cases} 1 & \text{if} \quad \displaystyle\sum_{i=1}^{3} x_i^2 < r^2, \\ -1 & \text{else.} \end{cases}$$

Here, $x \in [-1, 1]^{20}$ and x_i is the ith component of x. The radius $r = 0.125$. This test case was first proposed in [18]. Note that the separating surface here is still a 19-dimensional manifold.

In this example, we evaluate the classification error at 1000 random points uniformly sampled over the 20-dimensional subregion located within a distance 0.125 of the separating surface. If the 1000 uniformly random points instead covered the full 20-dimensional hypercube, the classification error would be excessively low; focusing on the region near the discontinuity, on the other hand, provides a more stringent test of our approximation scheme. To compare our algorithm's performance with that of the edge tracking scheme in [18], US iterations are continued until we achieve 93% classification error in the subregion. We use the same algorithm input parameters as in the previous example, with the exception of seeking only one edge point in each dimension, starting with an initial point set $\mathcal{M}_0$ that contains only the origin. The edge tracking results indicated that $\mathcal{O}(10^4)$ function evaluations were required to achieve 93% accuracy. The new discontinuity detection approach, on the other hand, requires 275 function evaluations function evaluations in the initialization phase and approximately 200 function evaluations during US. Again, this improvement reflects the fact that f contains a very regular discontinuity shape, which can be approximated by the kernel SVM quite efficiently.

6. Conclusions. This paper has developed an efficient and flexible discontinuity localization algorithm for the outputs of parameter-dependent computational simula-

tions. The algorithm progressively refines a functional approximation of the surface across which the discontinuity occurs. The approach is unstructured; after an initialization phase employing PA, it relies on guided random sampling and thus avoids constructing a dense structured grid of model evaluations, either globally or in the vicinity of the discontinuity. The separating surface is represented by a kernel SVM classifier; this representation is quite flexible and is able to capture a wide range of discontinuity geometries over a range of dimensions. We demonstrate the approach on several model functions and benchmark ODE and PDE systems. Compared to previous approaches, it requires significantly fewer model evaluations to achieve a given level of accuracy.

The advantage of this algorithm is greatest when the complexity of the separating surface is low, so that it can be well approximated with few points. But the nonparametric representation employed by the SVM allows it to approximate surfaces ranging from linear (i.e., hyperplanes) to very complex (i.e., disconnected). By contrast, other unsupervised discontinuity localization schemes assume that the geometry of the separating surface necessitates some form of edge tracking (in a sense allowing maximum complexity, up to a point-spacing tolerance δ) or assume a particular parameterization of the surface. In return for the present flexibility, one must make some assumptions about how quickly function values near the separating surface vary relative to the local jump size. When little is known, one can make conservative choices of the variation radius δ_t and the edge tolerance δ, and the method in a sense reverts to a Monte Carlo edge tracking scheme.

While we have demonstrated the effectiveness of the algorithm on a wide variety of problems, future refinements can extend it to simulations whose input domains must be divided into more than two classes, perhaps resulting from several disconnected output regimes, and to simulations that exhibit discontinuities in their derivatives. Tackling these issues should not fundamentally change the present methodology; indeed, one could employ multiclass SVM classifiers with an appropriate labeling scheme. Finally, we emphasize that the localization of discontinuities is but one step toward the development of efficient surrogates for computational simulations. Future work will couple the methodology presented here with function approximation techniques to create a unified framework for the construction of piecewise smooth surrogate models.

REFERENCES

[1] R. Archibald, A. Gelb, R. Saxena, and D. Xiu, *Discontinuity detection in multivariate space for stochastic simulations*, J. Comput. Phys., 228 (2009), pp. 2676–2689.

[2] R. Archibald, A. Gelb, and J. Yoon, *Polynomial fitting for edge detection in irregularly sampled signals and images*, SIAM J. Numer. Anal., 43 (2006), pp. 259–279.

[3] A. Basudhar and S. Missoum, *Adaptive explicit decision functions for probabilistic design and optimization using support vector machines*, Computers & Structures, 86 (2008), pp. 1904–1917.

[4] I. Bilionis and N. Zabaras, *Multi-output local Gaussian process regression: Applications to uncertainty quantification*, J. Comput. Phys., 231 (2012), pp. 5718–5746.

[5] C. J. C. Burges, *A tutorial on support vector machines for pattern recognition*, Data Min. Knowl. Discov., 2 (1998), pp. 121–167.

[6] C. Chang and C. Lin, *LIBSUM: A library for support vector machines*, ACM Trans. Intell. Syst. Technol., 2 (2011), 27:1–27:27.

[7] T. Chantrasmi, A. Doostan, and G. Iaccarino, *Padé–Legendre approximants for uncertainty analysis with discontinuous response surfaces*, J. Comput. Phys., 228 (2009), pp. 7159–7180.

[8] D. Cohn, L. Atlas, and R. Ladner, *Improving generalization with active learning*, Machine Learning, 15 (1994), pp. 201–221.

[9] P. R. Conrad and Y. M. Marzouk, *Adaptive Smolyak pseudospectral approximations*, SIAM J. Sci. Comput., 35 (2013), pp. A2643–A2670.
[10] S. Conti and A. O'Hagan, *Bayesian emulation of complex multi-output and dynamic computer models*, J. Statist. Plann. Inference, 140 (2010), pp. 640–651.
[11] I. Daubechies, *Orthonormal bases of compactly supported wavelets*, Comm. Pure Appl. Math., 41 (1988), pp. 909–996.
[12] M. S. Eldred, C. G. Webster, and P. Constantine, *Evaluation of non-intrusive approaches for Wiener–Askey generalized polynomial chaos*, in Proceedings of the 10th AIAA Non-Deterministic Approaches Conference, Schaumburg, IL, 2008, p. 189.
[13] T. S. Gardner, C. R. Cantor, and J. J. Collins, *Construction of a genetic toggle switch in escherichia coli*, Nature, 403 (2000), pp. 339–342.
[14] R. Ghanem and P. D. Spanos, *Stochastic Finite Elements: A Spectral Approach*, Springer, New York, 1991.
[15] D. Ghosh and R. Ghanem, *Stochastic convergence acceleration through basis enrichment of polynomial chaos expansions*, Internat. J. Numer. Methods Engrg., 73 (2008), pp. 162–184.
[16] R. B. Gramacy and H. K. H. Lee, *Bayesian treed Gaussian process models with an application to computer modeling*, J. Amer. Statist. Assoc., 103 (2008). pp. 1119–1130.
[17] R. B. Gramacy, H. K. H. Lee, and W. G. Macready, *Parameter space exploration with Gaussian process trees*, in Proceedings of the 21st International Conference on Machine Learning, ACM, 2004, p. 45.
[18] J. D. Jakeman, R. Archibald, and D. Xiu, *Characterization of discontinuities in high-dimensional stochastic problems on adaptive sparse grids*, J. Comput. Phys., 230 (2011), pp. 3977–3997.
[19] J. D. Jakeman, A. Narayan, and D. Xiu, *Minimal multi-element stochastic collocation for uncertainty quantification of discontinuous functions*, J. Comput. Phys., 242 (2013), pp. 790–808.
[20] N. Kingsbury, *Image processing with complex wavelets*, Philos. Trans. Roy. Soc. Lond. Ser. A, 357 (1999), pp. 2543–2560.
[21] O. P. Le Maître, O. M. Knio, H. N. Najm, and R. Ghanem, *Uncertainty propagation using Wiener–Haar expansions*, J. Comput. Phys., 197 (2004), pp. 28–57.
[22] O. P. Le Maitre, H. N. Najm, R. Ghanem, and O. M. Knio, *Multi-resolution analysis of Wiener-type uncertainty propagation schemes*, J. Comput. Phys., 197 (2004), pp. 502–531.
[23] D. D. Lewis and J Catlett, *Heterogeneous Uncertainty Sampling for Supervised Learning*, Morgan Kaufmann, San Francisco, CA, 1994, pp. 148–156.
[24] Y. M. Marzouk and D. Xiu, *A stochastic collocation approach to Bayesian inference in inverse problems*, Commun. Comput. Phys., 6 (2009), pp. 826–847.
[25] J. Mercer, *Functions of positive and negative type, and their connection with the theory of integral equations*, Philos. Trans. Roy. Soc. Lond. Ser. A., 209 (1909), pp. 415–446.
[26] Y. Meyer, *Wavelets - Algorithms and Applications*, SIAM, Philadelphia, 1993.
[27] T. A. Moselhy and Y. M. Marzouk, *Bayesian inference with optimal maps*, J. Comput. Phys., 231 (2012), pp. 7815–7850.
[28] H. N. Najm, B. J. Debusschere, Y. M. Marzouk, S. Widmer, and O. P. Le Maître, *Uncertainty quantification in chemical systems*, Internat. J. Numer. Methods Engrg., 80 (2009), pp. 789–814.
[29] A. Narayan and D. Xiu, *Stochastic collocation methods on unstructured grids in high dimensions via interpolation*, SIAM J. Sci. Comput., 34 (2012), pp. A1729–A1752.
[30] J. C. Platt, *Fast training of support vector machines using sequential minimal optimization*, in Advances in Kernel Methods—Support Vector Learning, B. Schölkopf, C. J. C Burges, and A. J. Smola, eds., MIT Press, Cambridge, MA, 1998, pp. 185–208.
[31] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, MIT Press, Cambridge, MA, 2006.
[32] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn, *Design and analysis of computer experiments*, Statist. Sci., 4 (1989), pp. 409–423.
[33] K. Sargsyan, C. Safta, B. Debusschere, and H. N. Najm, *Uncertainty quantification given discontinuous model response and a limited number of model runs*, SIAM J. Sci. Comput., 34 (2012), pp. B44–B64.
[34] G. Schohn and D. Cohn, *Less is more: Active learning with support vector machines*, in Proceedings of the 17th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, 2000, pp. 839–846.
[35] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, 2002.
[36] B. Settles, *Active Learning Literature Survey*, Sciences, Technical report 1648, Univerisity of Wisconsin-Madison, Madison, WI, 2010.

[37] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Springer, New York, 1995.
[38] X. Wan and G. Karniadakis, *An adaptive multi-element generalized polynomial chaos method for stochastic differential equations*, J. Comput. Phys., 209 (2005), pp. 617–642.
[39] M. Webster, J. Scott, A. Sokolov, and P. Stone, *Estimating probability distributions from complex models with bifurcations: The case of ocean circulation collapse*, J. Environ. Systems, 31 (2007), pp. 1–21.
[40] D. Xiu, *Efficient collocational approach for parametric uncertainty analysis*, Comm. Comput. Phys., 2 (2007), pp. 293–309.
[41] D. Xiu, *Numerical Methods for Stochastic Computations: A Spectral Method Approach*, Princeton University Press, Princeton, NJ, 2010.
[42] D. Xiu and J. S. Hesthaven, *High-order collocation methods for differential equations with random inputs*, SIAM J. Sci. Comput., 27 (2005), pp. 1118–1139.